ICOT Technical Report: TR-019

TR-019

LFG in Prolog
—Toward a formal system for representing grammatical relations—

by
Hideki Yasukawa

August, 1983

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

LFG in prolog

- Toward a formal system for representing grammatical relations -

Hideki YASUKAWA
Institute for New Generation Computer Technology (ICOT)
Tokyo, 108, Japan

1. Introduction

Currently we are planning the text understanding system which reads the sentences in a textbook and answers the questions. This system is one example of the applications of natural language understanding system whose main focus is discourse understanding. In the first step of the designing the system, the design of the syntactic analysis modules are currently carried. This paper describes the preliminary implementation of the represention for grammatical relations.

Our approach to syntactic analysis is based on logic programming. The advantage of using logic programming language Prolog, [Pereira,et.al. 78], for natural language processing is often argued. The reason for that is:

(1) its fitness for the logical approach, for example Montague Grammar, etc. .

(2) its 2-way interpretation, i.e. declarative and procedural.

(3) its great expressive power.

Moreover, a parser called DCG is available in the DEC-10 prolog system [Pereira,Warren 80]. The main feature of DCG is :

(1) clear correspondence between definite clause and context free rule (CF rule).

(2) efficiency of the parser.

We use the grammar rule written in the form used in DCG, say DCG form, but our parsing system, called BUP system [Matsumoto,et.al. 83], uses a bottom-up parsing strategy in contrast to DCG. DCG form has clear correspondence to CF rule as grammar rule, and moreover the context sensitiveness of the grammar and the semantic analysis are described by prolog program

associated the DCG form grammar rule. DCG form is considered as a very powerful tool for syntactic analysis.

The fundamental aims for a syntactic analysis are to characterize as follows :

(1) to check the grammaticality conditions.

(2) to clarify the mapping between semantic structures and surface word and phrase configurations.

These analysis is performed by the DCG form grammar rule and its associated prolog programs. In spite of the descriptive power of DCG form, the grammar will become too complicated and too hard to understand. It is due to the fact that the arbitrary prolog programs can be associated with a grammar. A lot of problems, difficulties of debugging a grammar or maintaining a grammar or making changes to a grammar, e.t.c., will arise as in the field of software engineering. To avoid these problems, the formal system for representing grammatical relations is needed.

From another point of view, the need for the formal system for representing grammatical relations arises. It is the case when the designer of the grammar is not acquainted with prolog. For example, as is often the case, a linguist designs a grammar. In such situation, the higher level grammar description language based on CF rule should be offered to the user. In order to design a higher level grammar description language, the system for representing grammatical relations should be formally defined.

From the needs for the formal system for representing grammatical relations described above, we tried to implement it on DCG form. The point is to extract the description primitives used to represent the grammatical relations. The requirements for the primitives are:

(1) ability for simple description

(2) non redundant set of primitives

(3) sufficient descriptive power

(4) etc.

As the current linguistic theory which satisfies the above requirement, we adopt Lexical Functional Grammar (LFG) for our preliminary version of the formal system [Kaplan,Bresnan 82]. In

this paper, we describe the implementation of the primitives of LFG in Prolog and the introduction of LFG in BUP system.

2. Simple overview of LFG

In this section, the overview of LFG only necessary for understanding the rest of this paper is described. See [Kaplan,Bresnan 82] for details.

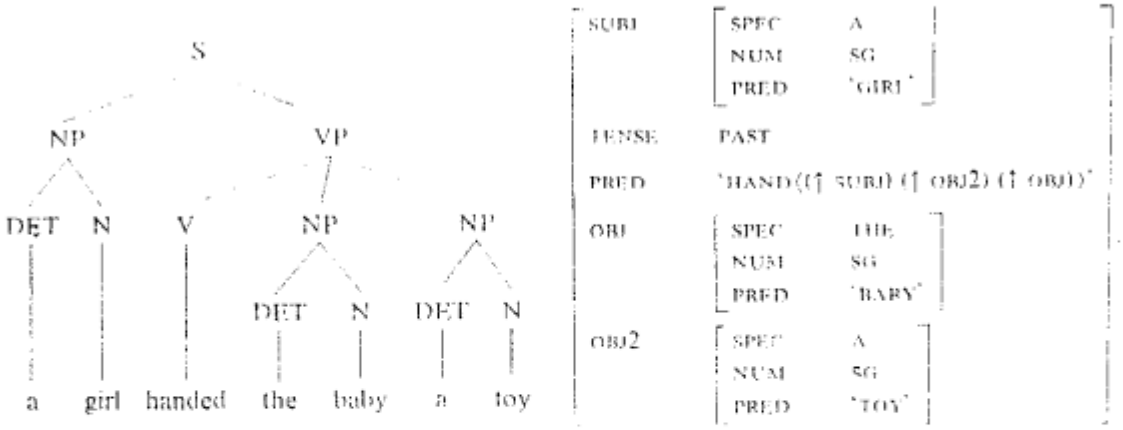
The major feature of LFG is :

(1) Lexical Grammar

(2) Two-level Representation (f-structure and c-structure)

In LFG, lexical entries specify a direct mapping between semantic arguments and configurations of surface grammatical functions. On the other hand, grammar rules specify a direct mapping between these surface grammatical functions and particular morphological and constituent structure configurations. Each of the two-level representations, f-structure (functional structure) and c-structure (constituent structure), represents the superficial arrangements of words and phrases in sentences and the configuration of the surface grammatical functions.

Fig. 1 shows the c-structure and f-structure for the example sentence "a girl handed the baby a toy".



(a) c-structure (b) f-structure

Fig. 1 The c-structure and f-structure for "a girl handed the baby a toy". ([Kaplan,Bresnan 82])

As shown in Fig. 1, f-structure is a hierarchical structure constructed by the pairs of attributes and their unique values. The attributes represent grammatical functions or syntactic features.

It is this f-structure that LFG seems to be appropriate for natural language processing application. To represent these grammatical relations, several devices are provided in LFG. These are :

(a) meta variables

(i) $\uparrow$ & $\downarrow$ immediate dominance
^ represent f-structure of a mother node
v represent f-structure of a daughter node

(ii) $\Uparrow$ & $\Downarrow$ bounded dominance
represent f-structure of the controllee
represent f-structure of the controler

(b) functional notations

For example, ($\uparrow$ subj) indicates the value of the attribute subj of the mother node's f-structure.

(c) Definitional Scheme

(i) = (equation)
specifies the value of a attribute of a f-structure or defines the relation of f-structures.

(ii) $\in$ (set inclusion)
defining the set constructs for f-structure.

(d) Constraining Scheme

(i) =c (equational constraints)
constraints for Definitional Scheme

(ii) d (existential constraint)
d is a designator such as ($\uparrow$ subj). constraints for the existence of the value for the designator.

(iii) ~ (negation)
symbol for describing negative constraints.

Fig. 2 shows the example of the grammar rules and lexical entries in LFG, which generate the c-structure and the f-structure in Fig. 1.

S → NP VP
(↑ SUBJ)=↓ ↑=↓

NP → DET N

VP → V NP NP
(↑ OBJ)=↓ (↑ OBJ2)=↓

a: DET, (↑ SPEC) = A
(↑ NUM) = SG

girl: N, (↑ NUM) = SG
(↑ PRED) = 'GIRL'

handed: V, (↑ TENSE) = PAST
(↑ PRED) = 'HAND((↑ SUBJ) (↑ OBJ2) (↑ OBJ))'

the: DET, (↑ SPEC) = THE

baby: N, (↑ NUM) = SG
(↑ PRED) = 'BABY'

toy: N, (↑ NUM) = SG
(↑ PRED) = 'TOY'

Fig. 2 Example grammar rules and lexical entries of LFG. ([Kaplan,Bresnan 82])

As shown in Fig. 2, the primitives representing grammatical relations are encoded in grammar rules and lexical entries. Attaching the grammatical and lexical schemata in Fig. 2 to the c-structure for sentence "a girl handed the baby a toy" produces the structure in Fig. 3.

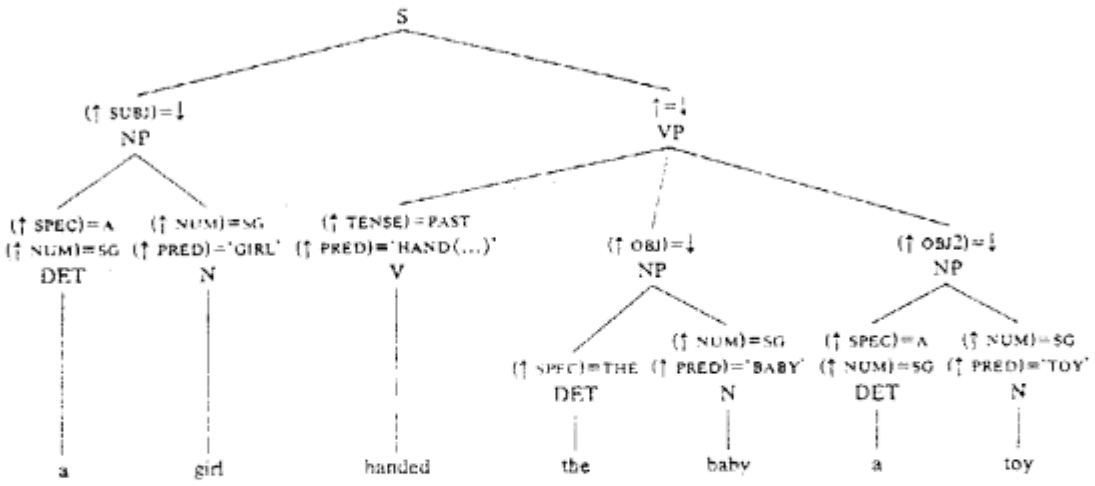


Fig. 3 The produced structure by attaching the grammatical and lexical schemata. ([Kaplan,Bresnan 82])

Definitional Scheme specifies the partial value of a f-structute in the course of the syntactic analysis. For example, the equation associated with the daughter node of the s in Fig. 3 only specifies that the value of the subj attribute of the f-structure of s is the f-structure of its daughter np. This specification is represented as ($\uparrow$ subj) = $\downarrow$, where $\uparrow$ and $\downarrow$ represents the f-structures of s and its daughter np each other and designater ($\uparrow$ subj) indicates the value of the subj attribute of the f-structure of s. But it does not say about the value assignments of the f-structure of np. If the syntactic analysis is performed in a bottom-up way, the value of the f-structure of np would be specified at that time, and if the top-down analysis is adopted, then the value of the f-structure of np would be undefined yet. Of course, the uniqueness of the value assignments, called Uniqueness Condition, is held, and it affects the grammaticality of a sentence. Constraining Scheme is the constraint for the values specified by Definitional Scheme. But Definitional Scheme and Constraining Scheme are independent each other. Constraining Scheme does not affect the control flow of Definitional Scheme, it only affects the value assignment of Definitional Scheme. But they have close relation for each other. The interaction between Definitional and Constraining Scheme are needed for an efficient implementation of LFG. Moreover both scheme are independent of the parsing process.

3. On Implementation of the primitives

As indicated in section 2, two distinct scheme is employed in the representation system in LFG. Those are :

(1) Definitional Scheme (equation, set inclusion)

(2) Constraining Scheme (equational or existential constraint)

Definitional Scheme plays a constructive role, in which partially specified f-structures are generated and modified. On the other hand, Constraining Scheme play a role of controling Definitional Scheme by checking the constraints associated with the particular equation, which are described in a grammar rule.

As in the case of introducing more general equality in prolog, implementation of equation of LFG results in :

(1) introducing newly defined data type

(2) introducing partially specified data object

(3) introducing functional notation

These features are adopted as the major principle of the implementation. Notice that the primitives for long distance dependence are not currently implemented.

3-1. Representations of data types

The primitive data types constructing f-structure is symbols, semantic predicates, subsidiary f-structures, and sets of symbols, semantic predicates, or f-structures. In current implementation, these data types are shown below.

1) symbols --> atom or integer

2) semantic predicates --> sem(X) where X is a predicate

3) f-structure --> Id:List where Id is an identifier variable which indicates the node of the parsing tree (syntactic category) having the f-structure, and List is a list of pairs of attribute and its value.

4) set --> {element1, element2, ..., elementN}

And the fifth data type, place-holder, is introduced in the implementation. It represents a f-structure or the value of an attribute of a f-structure whose value is undefined. It is introduced to describe the empty cell which represents the referenced designater whose value is undefined.

5) place-holder --> Id:Var where Id is an identifier variable as in f-structure. Var indicates a empty cell for the f-structure associated with the node identified with Id.

For example, the f-structure for the fragment of a sentence, "the baby" is represented internally as follows :

Id:[[num,sg],[per,3],[pred,sem(baby)],[spec,the]]

3-2 F-structure as partially specified data

As stated above, f-structure can be seen as a kind of partially specified data. In current Prolog, there is no direct way to represent partially specified data object. But specifying the partial value of the data is not a destructive assignment which is treated as side-effect. It is desirable to introduce the basic construct for representing such object. Currently the partial definition of a f-structure is treated as the side-effect of the definitional equations. And the side-effect is propagated by using two assignment list. The assignment list represent the correction of the data and variable assignments. The one of the two list represents the assignments before performing the definitional equations ($=, \in$) for a syntactic category, and the another represents the assignments after performing definitional equations.

The other approach to represent partially specified data object (f-structure) is being carried. These approaches are based on the use of difference list or ordered binary tree for assignment.

3-3 Functional notation

Meta variables $\uparrow$ and $\downarrow$ are represented by the variables associated with each syntactic category. These variables are used as the identifier variable of f-structures or place-holders. For example, if the designator ($\uparrow$ subj) is associated with the category , then it is represented as [subj, S] in current implementation. Notice that the left associativity of the function application is not currently available. So the designator ($\uparrow$ vcomp subj) is represented as [subj,[vcomp,S]].

3-4 Predicates for LFG primitives

The representation for each primitives in LFG is as follows : (d,d1,d2 are designator, s is a set, and ~ is negation symbol)

1) d1 = d2 -> equate(d1,d2,Old,New)

2) d $\in$ s -> include(d,s,Old,New)

3) d1 =c d2 -> constrain(d1,d2,OldC,NewC)

4) d (existential constraint) -> exist(d,OldC,NewC)

5) ~(d1 =c d2) -> neg_constrain(d1,d2,OldC,NewC)

6) ~d -> not_exist(d,OldC,NewC)

and the predicates for checking the constraints at the time when user specifies are provided. These predicates are needed form efficiency of the analysis. As the constraints are all satisfied in the final configuration of the f-structures without considering the analysis process, it is needed that the process generating inconsistent result is terminated as soon as the inconsistency is found in the process. Those predicates are :

7) check_constraint(Constraint,Assignment)

8) check_neg_constraint(Constraint,Assignment)

9) check_existence(Existential_constraint,Assignment)

10) check_inexistence(Existential_constraint,Assignment)

The Old and New above are the lists propagating the changes of assignments described in 3-2. The OldC and NewC is a list of the collections of the constraints appeared in the analysis of sentence, and is used in the similar way as Old and New.

The details of implementation is not main concern of this paper, but the realization of equate is briefly explained below. The main procedures of equate are locate and merge. equate is represented in terms of locate and merge as shown below. (Assignment lists are omitted.)

```
equate(d1,d2) -> merge(locate(d1),locate(d1))
```

locate gets the temporal value of the designater given as its argument from the assignment. merge 'merges' the values of the two designaters. Here, the meaning of the term 'merges' can be understood as 'minimally satisfied union of the values'. Fig. 4 shows some results of the trace of predicate equate in the course of analyzing the sentence "a girl hands the baby a toy".

```
***** Trace Start *****

Designator [spec,Det] locates
_465:_466                          ; indicates place-holder
Designator a locates
a

Result of merging is
a                                  ; variable _466 is
                                     assigned a value a
```

(a) Trace for equate([spec,Det],a,Old,New)

```
***** Trace Start *****

Designator Np locates
Np:_672                          ; value is not defined
                                   yet
Designator Det locates
Det:[[num,sg],[spec,a]]

Result of merging is
Det:[[num,sg],[spec,a]]          ; variable _672 is assigned a
                                   value [[num,sg],[spec,a]],and
                                   Np becomes equal to Det.
```

(b) Trace for equate(Np,Det,Old,New)

```
***** Trace Start *****

Designator Det locates
Det:[[num,sg],[spec,a]]
Designator N locates
N:[[pred,sem(girl)],[per,3],[num,sg]]

Result of merging is
Det:[[spec,a],[pred,sem(girl)],[per,3],[num,sg]]
                                 ; union of the two values are
                                   computed, and N becomes equal
                                   to Det.
```

(c) Trace for equate(Det,N,Old,New)

```
***** Trace Start *****

Designator [obj2,Vp] locates
_4160:_4161                      ; indicates place-holder
Designator Np locates
Np:[[per,3],[pred,sem(toy)],[num,sg],[spec,a]]

Result of merging is
Np:[[per,3],[pred,sem(toy)],[num,sg],[spec,a]]
                                 ; place-holder _4160:_4161
                                   becomes identical to the value
                                   of Np.
```

(d) Trace for equate([obj2,Vp],Np,Old,New)

Fig. 4 Tracing results of equate.

In Fig. 4, the results of locate of two designator is shown first, and then the result of merge of them is shown. The temporal assignments for each equation is omitted. As described above, these changes of f-structure are propagated by the two assignment list, Old and New.

3-5. Comment for current implementation

The use of identifier variable enables to get high efficiency about both speed and memory storage. The updates of the f-structures modified by equations are done only by the unifications of the identifier variables, and moreover, due to the inherent nature of the implementation of DEC-10 Prolog, i.e. structure sharing method, a f-structure can be shared from several upper level f-structures. The most time consumable process of equation is to search the entity corresponding to the designator from the assignment list. In current implementation, the redundant entities and variable assignments are removed from the assignment list.

The major problems with current implementation is:

1) the treatment of partially defined data.

2) the separation of the control scheme for constraint from Definitional Scheme.

3) the lack of the bounded domination meta variable.

The introduction of new data structure, such as ordered binary tree or difference list, may have good deal for the problem 1. The problem 2 affects the fundamental performance of the implementation. The introduction of parallel execution mechanism or concurrent mechanism should be considered.

The example shown below represents the prolog program that performs the analysis of the c-structure and the f-structure corresponding to the grammar rule.

```
s ->    np          vp
      (↑ subj)     ↑=↓
```

(a) The grammar rules in LFG notation

```
s(s(Np,Vp),Id_S,Old,New,OldC,NewC) -->
            np(Np,Id_Np,Old,Old1,OldC,OldC1),
            {equate([subj,Id_S],Id_Np,Old1,Old2)},
            vp(Vp,Id_Vp,Old2,Old3,OldC1,NewC),
```

```
{equate(Id_S,Id_Vp,Old3,New)}.
```

(b) Corresponding Prolog program

Fig. 5 Example Prolog program of LFG analysis

The variables Id_S, Id_Np, and Id_Vp is the identifier variable for each syntactic category. The variables such as Old, New, OldC, NewC, etc. , are the lists representing assignments and constraints.

4. Introduction of LFG primitives in BUP system

The Prolog implementation of LFG primitives described above is introduced to our parsing system BUP [Matsumoto,et.al. 83]. The aim of that is :

1) The method for avoiding redundant computation proposed in [Matsumoto,et.al 83] is available. The considerable improvement are expected by the method.
   As BUP's parsing strategy is bottom up, left recursive rule is available and the separation of the grammar rules and dictionary items can be done.

2) As shown in Fig. 5, direct introduction of LFG primitives in DCG is not desirable for the user. There is no need to let the user write down the assignment list or constraint list, and
   moreover it would be better to hide such unnecessary data from the user for keeping the transparency of the grammar.
   On introducing the LFG primitives into BUP, it is achieved by describing the grammatical relations similar to LFG using macro notations. BUP translater translates these descriptions into Prolog programs of LFG primitives, i.e. equate, include, constrain, or etc. . This macro expansion results in considerable improvement of the descriptive power and the ease for understanding of grammar.

The macro notation of the LFG primitives stated in 2) above is associated with the predicate which corresponds to the syntactic category as its argument. The syntax of macro notations are :

```
(a) d1 = d2 -> eq(d1,d2),
        which is translated into equate(d1,d2,Old,New)

(b) d ∈ s -> incl(d,s),
```

```
        which is translated into include(d1,s,Old,New)

(c) d1 =c d2 -> c(d1,d2),
        which is translated into constrain(d1,d2,Old,New)

(d) d -> ex(d),
        which is translated into exist(d,Old,New)

(e) ~(d1 =c d2) -> not_c(d1,d2),
        which is translated into neg_constrain(d1,d2,Old,New)

(f) ~d -> not_ex(d),
        which is translated into not_exist(d,Old,New)

(g) check_c(c),
        which is translated into check_constraint(c,Assign)

(h) check_not_c(c),
        which is translated into check_neg_constraint(c,Assign)

(i) check_ex(d),
        which is translated into check_existence(d,Assign)

(j) check_not_ex(d),
        which is translated into check_inexistence(d,Assign)
```

These macro notation of LFG primitives are associated with the predicate corresponding to each syntactic category as their third argument. For example, the grammar rule in Fig. 5 is represented in BUP version of LFG system as in Fig. 6 below.

```
s(s(Np,Vp),Id_S,[]) --> np(Np,Id_Np,[eq([subj,Id_S],Id_Np]),
                        vp(Vp,Id_Vp,[eq(Id_S,Id_Vp)]).
```

Fig. 6 Example grammar rule of BUP

The rule in Fig. 6 is translated into the Prolog program which performs the analysis of corresponding LFG rule with a bottom-up parsing strategy. The program is shown in Fig. 7.

```
np(_B,[Np,Id_Np],_C,_D,_E,_F,_G,_H,_I) :-
        link(s,_B),
        equate([subj,Id_S],Np,_E,_J),
        goal(vp,[Vp,Id_Vp],_C,_K,_J,_L,_G,_M),
        equate(Id_S,Id_Vp,_L,_N),
        call(s(_B,[s(Np,Vp),Id_S],_K,_D,_N,_F,_M,_H,_I)).
```

Fig. 7 Prolog program corresponds to the grammar rule in Fig. 5

In Appendix 1, a simple LFG grammar for BUP system is shown. And the resulting Prolog programs of translating it is shown in Appendix 2.

5. The result of some experiments

Fig. 8 shows the result of analyzing the sentence "a girl persuaded the baby to go".

```
| ?- parse.

LFG System (in BUP) Start. Please Input Sentence.

|: a girl persuaded the baby to go.


Time used in analysis are
     1186 ms. for syntactic analysis
     16 ms. for checking constraints

The result of the analysis is as follows

 |-s
    |-np
    |  |-det
    |  |  |-a
    |  |-n
    |     |-girl
    |-vp
       |-v
       |  |-persuaded
       |-np
       |  |-det
       |  |  |-the
       |  |-n
       |     |-baby
       |-vpcomp
          |-to
          |-vp
             |-v
                |-go

 Assignments for category s is

    _1405:
          [num,sg]
```

```
      [per,3]
      [pred,sem(baby)]
      [spec,the]
_1416:
      [inf,+]
      [pred,sem(go(subj))]
      [to,+]
      [subj,_1405]
_43:
      [tense,past]
      [pred,sem(persuade(subj,obj,vcomp))]
      [vcomp,_1416]
      [obj,_1405]
      [subj,_576]
_576:
      [per,3]
      [pred,sem(girl)]
      [num,sg]
      [spec,a]
```

Fig. 8 The result of analyzing the sentence, "a girl persuaded the baby to go"

The underline prefixed numbers, for example _43, are identifier variables. In above example, _43 is the identifier variables for category s, _576 is the one for category np immediately dominated by s, i.e. a girl, _1405 is the identifier variable for category np immediately dominated by vp, i.e. the baby, and _1416 is the identifier variable for category vp. The functional control [Kaplan,Bresnan 82] is realized as shown in the f-structure for vp. The value of attribute subj of f-structure for vp is the f-structure for np immediately dominated by vp, i.e. the baby, while the one in (e) of Appendix 3, i.e. the result of analyzing "a girl promised the baby to go", is the f-structure for np immediately dominated by s.

The time used for syntactic analysis includes the time consumed by bottom up parsing process and the time consumed by Definitional Scheme.

Appendix 3 shows the other results of the analysis of LFG rules shown in Appendix 1. As the grammar is too small to get general conclusion, the approach to syntactic analysis described in this paper seems to be useful from the time used in a analysis. I believe that this kind of approach to syntax is desirable tool for natural language processing system. And as stated in 3-2, to introduce new control strategy to get more powerful and efficient result.

The major problem is extracting the more primitives necessary for treating the variety of utterences appeared in usual discourse situation, and giving the necessary scheme for those primitives. LFG indicates the way to the problem, but more experiments on

actual syntactic analysis on BUP system must be done for the satisfiable system for representing grammatical relations. For the first step, I try to introduce the primitives for long distance dependency and to extend the syntax of the constraint. This will lead to introducing the new control strategy such as concurrent execution of Prolog, or coroutining by bind hook mechanism.

6. Conclusion

This paper describes the implementation of LFG in Prolog, which is the first step of our formal system for representing grammatical relations. Further research on the formal system will be carried by analyzing the wider variety of statements. And the formal system will be used as a kernel system of the syntactic analysis of our text understanding system. The author would like to thank Dr. Kouichi Furukawa, Chief of the Second Research Laboratory, for his valuable daily guidance. And The author also thank the members of Natural Language Processing Group of the Second Research Laboratory.

<References>

[Pereira,et.al. 78] "User's Guide to DEC System-10 Prolog", Department of Artificial Intelligence, Univ. of Edinburgh, 1978

[Pereira,Warren 80] "Definite Clause Grammar for Language Analysis -- A Survey of the Formalism and a Comparison with Augmented Transition Networks", Artificial Intelligence, 13, pp. 231-278 , 1980

[Matsumoto,et.al. 83] "BUP: A Bottom-Up Parser embedded in Prolog", To appear in New Generation Computing, Vol. 2, OHM-Springer, 1983

[Kaplan,Bresnan 82] "Lexical-Functional Grammar: A Formal System for Grammatical Representation" in "Mental Representation of Grammatical Relations", Bresnan eds., MIT Press, 1982

Appendix 1. Example grammar in DCG form

```
/* grammars */

s(s(Stnp,Stvp),S,[]) -->
   np(Stnp,Np,[eq([subj,S],Np)]),
   vp(Stvp,Vp,[eq(S,Vp)]).


np(np(Stdet,Stn),Np,[]) -->
   det(Stdet,Det,[eq(Np,Det)]),
   n(Stn,N,[eq(Np,N)]).

vp(vp(Stv),Vp,[]) -->
   v(Stv,V,[eq(Vp,V)]).

vp(vp(Stv,Stnp1,Stnp2),Vp,[]) -->
   v(Stv,V,[eq(Vp,V)]),
   np(Stnp1,Np1,[eq([obj,Vp],Np1)]),
   np(Stnp2,Np2,[eq([obj2,Vp],Np2)]).

vp(vp(Stv,Stnp1,Stpp),Vp,[]) -->
   v(Stv,V,[eq(Vp,V)]),
   np(Stnp1,Np1,[eq([obj,Vp],Np1)]),
   pp(Stpp,Pp,[eq([[pcase,Pp],Vp],Pp)]).

vp(vp(Stv,Stnp1,Stpp,Stpp1),Vp,[]) -->
   v(Stv,V,[eq(Vp,V)]),
   np(Stnp1,Np1,[eq([obj,Vp],Np1)]),
   pp(Stpp,Pp,[eq([[pcase,Pp],Vp],Pp)]),
   pp(Stpp1,Pp1,[incl(Pp1,[adjunct,Vp])]).

vp(vp(Stv,Stnp,Stvpcomp),Vp,[]) -->
   v(Stv,V,[eq(Vp,V)]),
   np(Stnp,Np,[eq([obj,Vp],Np)]),
   vpcomp(Stvpcomp,Vpcomp,[eq([vcomp,Vp],Vpcomp)]).

vp(vp(Stv,Stvpcomp),Vp,[]) -->
   v(Stv,V,[eq(Vp,V)]),
   vpcomp(Stvpcomp,Vpcomp,[eq([vcomp,Vp],Vpcomp)]).


vpcomp(vpcomp(Stvp),Vpcomp,[]) -->
   vp(Stvp,Vp,[eq(Vpcomp,Vp)]).

vpcomp(vpcomp(to,Stvp),Vpcomp,
       [eq([to,Vpcomp],+),c([inf,Vpcomp],+)]) -->
   [to],
   vp(Stvp,Vp,[eq(Vpcomp,Vp)]).
```

```
pp(pp(Stp,Stnp),Pp,[]) -->
   p(Stp,P,[eq(Pp,P)]),
   np(Stnp,Np,[eq([obj,Pp],Np)]).

/* dictionaries */

det(det(a),Det,
    [eq([spec,Det],a),
     eq([num,Det],sg)]) -->
   [a].

det(det(the),Det,
    [eq([spec,Det],the)]) -->
   [the].

n(n(girl),N,
  [eq([num,N],sg),
   eq([per,N],3),
   eq([pred,N],sem(girl))]) -->
   [girl].

n(n(girls),N,
  [eq([num,N],pl),
   eq([pred,N],sem(girl))]) -->
   [girls].

n(n(baby),N,
  [eq([num,N],sg),
   eq([per,N],3),
   eq([pred,N],sem(baby))]) -->
   [baby].

n(n(toy),N,
  [eq([num,N],sg),
   eq([per,N],3),
   eq([pred,N],sem(toy))]) -->
   [toy].

n(n(morning),N,
  [eq([num,N],sg),
   eq([per,N],3),
   eq([pred,N],sem(morning))]) -->
   [morning].

n(n(room),N,
  [eq([num,N],sg),
   eq([per,N],3),
   eq([pred,N],sem(room))]) -->
   [room].

v(v(go),V,
  [eq([inf,V],+),
```

```
    eq([pred,V],sem(go(subj)))]) -->
  [go].

v(v(hand),V,
  [not_c([per,[subj,V]],3),
   eq([tense,V],present),
   eq([pred,V],sem(hand(subj,obj,obl)))]) -->
   [hand].

v(v(hands),V,
  [eq([tense,V],present),
   eq([num,[subj,V]],sg),
   eq([per,[subj,V]],3),
   eq([pred,V],sem(hand(subj,obj,obl)))]) -->
   [hands].

v(v(handing),V,
   [eq([participle,V],present),
    eq([pred,V],sem(hand(subj,obj,obl)))]) -->
   [handing].

v(v(is),V,
   [eq([tense,V],present),
    eq([num,[subj,V]],sg),
    eq([pred,V],sem(prog(vcomp))),
    c([participle,[vcomp,V]],present),
    eq([subj,[vcomp,V]],[subj,V])]) -->
   [is].

v(v(persuaded),V,
   [eq([tense,V],past),
    eq([pred,V],sem(persuade(subj,obj,vcomp))),
    c([to,[vcomp,V]],+),
    eq([subj,[vcomp,V]],[obj,V])]) -->
   [persuaded].

v(v(promised),V,
   [eq([tense,V],past),
    eq([pred,V],sem(promise(subj,obj,vcomp))),
    c([to,[vcomp,V]],+),
    eq([subj,[vcomp,V]],[subj,V])]) -->
   [promised].

p(p(to),P,[eq([pcase,P],to)]) -->
   [to].

p(p(in),P,[eq([pcase,P],in)]) -->
   [in].

p(p(on),P,[eq([pcase,P],on)]) -->
   [on].
```

```
p(p(at),P,[eq([pcase,P],at)]) -->
   [at].

p(p(with),P,[eq([pcase,P],with)]) -->
   [with].

p(p(for),P,[eq([pcase,P],for)]) -->
   [for].

p(p(of),P,[eq([pcase,P],of)]) -->
   [of].
```

Appendix 2. Translation result of the grammar in Appendix 1

```
:-mode dict(?,?,+,-,?,?,?,?).
:-public dict/8.
:-mode link(+,+).
:-public link/2.
:-mode goal(+,?,+,-,?,-,?,-).
:-public goal/8.
:-mode w_f_subgoal(+,?,+,-,?,-,?,-).
:-public w_f_subgoal/8.
:-mode p(+,?,+,?,?,?,?,?,?).
:-public p/9.
:-mode terminal1(+,?,+,?,?,?,?,?,?).
:-public terminal1/9.
:-mode vpcomp(+,?,+,?,?,?,?,?,?).
:-public vpcomp/9.
:-mode pp(+,?,+,?,?,?,?,?,?).
:-public pp/9.
:-mode v(+,?,+,?,?,?,?,?,?).
:-public v/9.
:-mode det(+,?,+,?,?,?,?,?,?).
:-public det/9.
:-mode n(+,?,+,?,?,?,?,?,?).
:-public n/9.
:-mode np(+,?,+,?,?,?,?,?,?).
:-public np/9.
:-mode vp(+,?,+,?,?,?,?,?,?).
:-public vp/9.
:-mode s(+,?,+,?,?,?,?,?,?).
:-public s/9.
link(X,X).
link(terminal1,vpcomp).
link(vp,vpcomp).
link(v,vpcomp).
link(p,pp).
link(det,np).
link(v,vp).
link(np,s).
link(det,s).
p(p,_A,_B,_B,_C,_C,_D,_D,_A).
terminal1(terminal1,_A,_B,_B,_C,_C,_D,_D,_A).
vpcomp(vpcomp,_A,_B,_B,_C,_C,_D,_D,_A).
pp(pp,_A,_B,_B,_C,_C,_D,_D,_A).
v(v,_A,_B,_B,_C,_C,_D,_D,_A).
det(det,_A,_B,_B,_C,_C,_D,_D,_A).
n(n,_A,_B,_B,_C,_C,_D,_D,_A).
np(np,_A,_B,_B,_C,_C,_D,_D,_A).
vp(vp,_A,_B,_B,_C,_C,_D,_D,_A).
s(s,_A,_B,_B,_C,_C,_D,_D,_A).
np(_B,[Stnp,Np],_C,_D,_E,_F,_G,_H,_I) :-
        link(s,_B),
```

```
        equate([subj,S],Np,_E,_J),
        goal(vp,[Stvp,Vp],_C,_K,_J,_L,_G,_M),
        equate(S,Vp,_L,_N),
        call(s(_B,[s(Stnp,Stvp),S],_K,_D,_N,_F,_M,_H,_I)).

det(_B,[Stdet,Det],_C,_D,_E,_F,_G,_H,_I) :-
        link(np,_B),
        equate(Np,Det,_E,_J),
        goal(n,[Stn,N],_C,_K,_J,_L,_G,_M),
        equate(Np,N,_L,_N),
        call(np(_B,[np(Stdet,Stn),Np],_K,_D,_N,_F,_M,_H,_I)).

v(_B,[Stv,V],_C,_D,_E,_F,_G,_H,_I) :-
        link(vp,_B),
        equate(Vp,V,_E,_J),
        call(vp(_B,[vp(Stv),Vp],_C,_D,_J,_F,_G,_H,_I)).

v(_B,[Stv,V],_C,_D,_E,_F,_G,_H,_I) :-
        link(vp,_B),
        equate(Vp,V,_E,_J),
        goal(np,[Stnp1,Np1],_C,_K,_J,_L,_G,_M),
        equate([obj,Vp],Np1,_L,_N),
        goal(np,[Stnp2,Np2],_K,_O,_N,_P,_M,_Q),
        equate([obj2,Vp],Np2,_P,_R),
        call(vp(_B,[vp(Stv,Stnp1,Stnp2),Vp],_O,_D,_R,_F,_Q,_H,_I)).

v(_B,[Stv,V],_C,_D,_E,_F,_G,_H,_I) :-
        link(vp,_B),
        equate(Vp,V,_E,_J),
        goal(np,[Stnp1,Np1],_C,_K,_J,_L,_G,_M),
        equate([obj,Vp],Np1,_L,_N),
        goal(pp,[Stpp,Pp],_K,_O,_N,_P,_M,_Q),
        equate([[pcase,Pp],Vp],Pp,_P,_R),
        call(vp(_B,[vp(Stv,Stnp1,Stpp),Vp],_O,_D,_R,_F,_Q,_H,_I)).

v(_B,[Stv,V],_C,_D,_E,_F,_G,_H,_I) :-
        link(vp,_B),
        equate(Vp,V,_E,_J),
        goal(np,[Stnp1,Np1],_C,_K,_J,_L,_G,_M),
        equate([obj,Vp],Np1,_L,_N),
        goal(pp,[Stpp,Pp],_K,_O,_N,_P,_M,_Q),
        equate([[pcase,Pp],Vp],Pp,_P,_R),
        goal(pp,[Stpp1,Pp1],_O,_S,_R,_T,_Q,_U),
        include(Pp1,[adjunct,Vp],_T,_V),
        call(vp(_B,[vp(Stv,Stnp1,Stpp,Stpp1),Vp],_S,_D,_V,_F,_U,_H,_I)).

v(_B,[Stv,V],_C,_D,_E,_F,_G,_H,_I) :-
        link(vp,_B),
        equate(Vp,V,_E,_J),
        goal(np,[Stnp,Np],_C,_K,_J,_L,_G,_M),
        equate([obj,Vp],Np,_L,_N),
        goal(vpcomp,[Stvpcomp,Vpcomp],_K,_O,_N,_P,_M,_Q),
```

```
        equate([vcomp,Vp],Vpcomp,_P,_R),
        call(vp(_B,[vp(Stv,Stnp,Stvpcomp),Vp],_O,_D,_R,_F,_Q,_H,_I)).

v(_B,[Stv,V],_C,_D,_E,_F,_G,_H,_I) :-
        link(vp,_B),
        equate(Vp,V,_E,_J),
        goal(vpcomp,[Stvpcomp,Vpcomp],_C,_K,_J,_L,_G,_M),
        equate([vcomp,Vp],Vpcomp,_L,_N),
        call(vp(_B,[vp(Stv,Stvpcomp),Vp],_K,_D,_N,_F,_M,_H,_I)).

vp(_B,[Stvp,Vp],_C,_D,_E,_F,_G,_H,_I) :-
        link(vpcomp,_B),
        equate(Vpcomp,Vp,_E,_J),
        call(vpcomp(_B,[vpcomp(Stvp),Vpcomp],_C,_D,_J,_F,_G,_H,_I)).

terminal1(_B,[_C,_D],_E,_F,_G,_H,_I,_J,_K) :-
        link(vpcomp,_B),
        goal(vp,[Stvp,Vp],_E,_L,_G,_M,_I,_N),
        equate(Vpcomp,Vp,_M,_O),
        call(vpcomp(_B,[vpcomp(to,Stvp),Vpcomp],_L,_F,_O,_P,_N,_Q,_K)),
        equate([to,Vpcomp],+,_P,_H),
        constrain([inf,Vpcomp],+,_Q,_J,_H).

dict(terminal1,[_A,_B],[to|_C],_C,_D,_D,_E,_E).
p(_B,[Stp,P],_C,_D,_E,_F,_G,_H,_I) :-
        link(pp,_B),
        equate(Pp,P,_E,_J),
        goal(np,[Stnp,Np],_C,_K,_J,_L,_G,_M),
        equate([obj,Pp],Np,_L,_N),
        call(pp(_B,[pp(Stp,Stnp),Pp],_K,_D,_N,_F,_M,_H,_I)).

dict(det,[det(a),Det],[a|_B],_B,_C,_D,_E,_E) :-
        equate([spec,Det],a,_C,_F),
        equate([num,Det],sg,_F,_D).

dict(det,[det(the),Det],[the|_B],_B,_C,_D,_E,_E) :-
        equate([spec,Det],the,_C,_D).

dict(n,[n(girl),N],[girl|_B],_B,_C,_D,_E,_E) :-
        equate([num,N],sg,_C,_F),
        equate([per,N],3,_F,_G),
        equate([pred,N],sem(girl),_G,_D).

dict(n,[n(girls),N],[girls|_B],_B,_C,_D,_E,_E) :-
        equate([num,N],pl,_C,_F),
        equate([pred,N],sem(girl),_F,_D).

dict(n,[n(baby),N],[baby|_B],_B,_C,_D,_E,_E) :-
        equate([num,N],sg,_C,_F),
        equate([per,N],3,_F,_G),
        equate([pred,N],sem(baby),_G,_D).
```

```
dict(n,[n(toy),N],[toy|_B],_B,_C,_D,_E,_E) :-
        equate([num,N],sg,_C,_F),
        equate([per,N],3,_F,_G),
        equate([pred,N],sem(toy),_G,_D).

dict(n,[n(morning),N],[morning|_B],_B,_C,_D,_E,_E) :-
        equate([num,N],sg,_C,_F),
        equate([per,N],3,_F,_G),
        equate([pred,N],sem(morning),_G,_D).

dict(n,[n(room),N],[room|_B],_B,_C,_D,_E,_E) :-
        equate([num,N],sg,_C,_F),
        equate([per,N],3,_F,_G),
        equate([pred,N],sem(room),_G,_D).

dict(v,[v(go),V],[go|_B],_B,_C,_D,_E,_E) :-
        equate([inf,V],+,_C,_F),
        equate([pred,V],sem(go(subj)),_F,_D).

dict(v,[v(hand),V],[hand|_B],_B,_C,_D,_E,_F) :-
        neg_constrain([per,[subj,V]],3,_E,_F,_C),
        equate([tense,V],present,_C,_G),
        equate([pred,V],sem(hand(subj,obj,obl)),_G,_D).

dict(v,[v(hands),V],[hands|_B],_B,_C,_D,_E,_E) :-
        equate([tense,V],present,_C,_F),
        equate([num,[subj,V]],sg,_F,_G),
        equate([per,[subj,V]],3,_G,_H),
        equate([pred,V],sem(hand(subj,obj,obl)),_H,_D).

dict(v,[v(handing),V],[handing|_B],_B,_C,_D,_E,_E) :-
        equate([participle,V],present,_C,_F),
        equate([pred,V],sem(hand(subj,obj,obl)),_F,_D).

dict(v,[v(is),V],[is|_B],_B,_C,_D,_E,_F) :-
        equate([tense,V],present,_C,_G),
        equate([num,[subj,V]],sg,_G,_H),
        equate([pred,V],sem(prog(vcomp)),_H,_I),
        constrain([participle,[vcomp,V]],present,_E,_F,_I),
        equate([subj,[vcomp,V]],[subj,V],_I,_D).

dict(v,[v(persuaded),V],[persuaded|_B],_B,_C,_D,_E,_F) :-
        equate([tense,V],past,_C,_G),
        equate([pred,V],sem(persuade(subj,obj,vcomp)),_G,_H),
        constrain([to,[vcomp,V]],+,_E,_F,_H),
        equate([subj,[vcomp,V]],[obj,V],_H,_D).

dict(v,[v(promised),V],[promised|_B],_B,_C,_D,_E,_F) :-
        equate([tense,V],past,_C,_G),
        equate([pred,V],sem(promise(subj,obj,vcomp)),_G,_H),
        constrain([to,[vcomp,V]],+,_E,_F,_H),
        equate([subj,[vcomp,V]],[subj,V],_H,_D).
```

```
dict(p,[p(to),P],[to|_B],_B,_C,_D,_E,_E) :-
        equate([pcase,P],to,_C,_D).

dict(p,[p(in),P],[in|_B],_B,_C,_D,_E,_E) :-
        equate([pcase,P],in,_C,_D).

dict(p,[p(on),P],[on|_B],_B,_C,_D,_E,_E) :-
        equate([pcase,P],on,_C,_D).

dict(p,[p(at),P],[at|_B],_B,_C,_D,_E,_E) :-
        equate([pcase,P],at,_C,_D).

dict(p,[p(with),P],[with|_B],_B,_C,_D,_E,_E) :-
        equate([pcase,P],with,_C,_D).

dict(p,[p(for),P],[for|_B],_B,_C,_D,_E,_E) :-
        equate([pcase,P],for,_C,_D).

dict(p,[p(of),P],[of|_B],_B,_C,_D,_E,_E) :-
        equate([pcase,P],of,_C,_D).

goal(CurGoal,Arg,S0,S,Ass0,Ass,C0,C) :-
        w_f_subgoal(CurGoal,_,S0,_,Ass0,_,C0,_),
        !,
        w_f_subgoal(CurGoal,Arg,S0,S,Ass0,Ass,C0,C).

goal(CurGoal,Arg,S0,S,Ass0,Ass,C0,C) :-
        dict(Nt,Arg1,S0,S1,Ass0,Ass1,C0,C1),
        link(Nt,CurGoal),
        Pred=..[Nt,CurGoal,Arg1,S1,S,Ass1,Ass,C1,C,Arg],
        Pred,
        asserta(w_f_subgoal(CurGoal,Arg,S0,S,Ass0,Ass,C0,C)).
```

Appendix 3. Some results of the analysis

```
LFG System (in BUP) Start. Please Input Sentence.

|: a girl hands the baby a toy.


Time used in analysis are
     1092 ms. for syntactic analysis
     0 ms. for checking constraints

The result of the analysis is as follows

 |-s
    |-np
    |  |-det
    |  |  |-a
    |  |-n
    |     |-girl
    |-vp
       |-v
       |  |-hands
       |-np
       |  |-det
       |  |  |-the
       |  |-n
       |     |-baby
       |-np
          |-det
          |  |-a
          |-n
             |-toy

Assignments for category s is

   _1410:
         [per,3]
         [pred,sem(toy)]
         [num,sg]
         [spec,a]
   _1408:
         [num,sg]
         [per,3]
         [pred,sem(baby)]
         [spec,the]
   _43:
         [tense,present]
         [pred,sem(hand(subj,obj,obl))]
         [obj,_1408]
         [obj2,_1410]
         [subj,_448]
```

```
_448:
      [per,3]
      [pred,sem(girl)]
      [num,sg]
      [spec,a]
```

(a) The result of analyzing "a girl hands the baby a toy"

```
LFG System (in BUP) Start. Please Input Sentence.

|: a girl hand the baby a toy.

no
| ?-
```

(b) The result of analyzing "a girl hand the baby a toy"

```
LFG System (in BUP) Start. Please Input Sentence.

|: a girl hands a toy to the baby.


Time used in analysis are
     1257 ms. for syntactic analysis
     1 ms. for checking constraints

The result of the analysis is as follows

 |-s
    |-np
    |  |-det
    |  |  |-a
    |  |-n
    |     |-girl
    |-vp
       |-v
       |  |-hands
       |-np
       |  |-det
       |  |  |-a
       |  |-n
       |     |-toy
       |-pp
          |-p
          |  |-to
          |-np
             |-det
             |  |-the
             |-n
                |-baby

 Assignments for category s is

    _1720:
          [num,sg]
          [per,3]
          [pred,sem(baby)]
          [spec,the]
    _1463:
          [obj,_1720]
          [pcase,to]
    _1461:
          [per,3]
          [pred,sem(toy)]
          [num,sg]
          [spec,a]
    _43:
          [tense,present]
          [pred,sem(hand(subj,obj,obl))]
```

```
      [obj,_1461]
      [to,_1463]
      [subj,_501]
_501:
      [per,3]
      [pred,sem(girl)]
      [num,sg]
      [spec,a]
```

(c) The result of analyzing "a girl hands a toy to the baby"

```
LFG System (in BUP) Start. Please Input Sentence.

|: a girl is handing a toy to the baby in the morning.


Time used in analysis are
     3691 ms. for syntactic analysis
     17 ms. for checking constraints

The result of the analysis is as follows

 |-s
    |-np
    |  |-det
    |  |  |-a
    |  |-n
    |     |-girl
    |-vp
       |-v
       |  |-is
       |-vpcomp
          |-vp
             |-v
             |  |-handing
             |-np
             |  |-det
             |  |  |-a
             |  |-n
             |     |-toy
             |-pp
             |  |-p
             |  |  |-to
             |  |-np
             |     |-det
             |     |  |-the
             |     |-n
             |        |-baby
             |-pp
                |-p
                |  |-in
                |-np
                   |-det
                   |  |-the
                   |-n
                      |-morning

 Assignments for category s is

    _3707:
             [
             _2177:
                   [obj,_2609]
```

```
                [pcase,in]
        }
_2609:
      [num,sg]
      [per,3]
      [pred,sem(morning)]
      [spec,the]
_2177:
      [obj,_2609]
      [pcase,in]
_2385:
      [num,sg]
      [per,3]
      [pred,sem(baby)]
      [spec,the]
_2175:
      [obj,_2385]
      [pcase,to]
_2173:
      [per,3]
      [pred,sem(toy)]
      [num,sg]
      [spec,a]
_1838:
      [participle,present]
      [pred,sem(hand(subj,obj,obl))]
      [obj,_2173]
      [to,_2175]
      [adjunct,_3707]
      [subj,_934]
_43:
      [tense,present]
      [pred,sem(prog(vcomp))]
      [vcomp,_1838]
      [subj,_934]
_934:
      [per,3]
      [pred,sem(girl)]
      [num,sg]
      [spec,a]
```

(d) The result of analyzing "a girl is handing a toy to the baby in the morning."

```
LFG System (in BUP) Start. Please Input Sentence.

|: a girl promised the baby to go.


Time used in analysis are
     1167 ms. for sysnatctic analysis
     16 ms. for checking constraints

The result of the analysis is as follows

 |-s
    |-np
    |  |-det
    |  |  |-a
    |  |-n
    |     |-girl
    |-vp
       |-v
       |  |-promised
       |-np
       |  |-det
       |  |  |-the
       |  |-n
       |     |-baby
       |-vpcomp
          |-to
          |-vp
             |-v
                |-go

Assignments for category s is

   _1512:
         [num,sg]
         [per,3]
         [pred,sem(baby)]
         [spec,the]
   _1377:
         [inf,+]
         [pred,sem(go(subj))]
         [to,+]
         [subj,_537]
   _43:
         [tense,past]
         [pred,sem(promise(subj,obj,vcomp))]
         [vcomp,_1377]
         [obj,_1512]
         [subj,_537]
   _537:
         [per,3]
         [pred,sem(girl)]
```

[num,sg]
[spec,a]

(e) The result of analyzing "a girl promised the baby to go"